Les Listes Chainées

1- Introduction :

Jusqu'à maintenant, nous avons vu la représentation des données sous forme contigüe (séquentielle) ; c'est-à-dire en utilisant des tableaux. Cette représentation est très adéquate pour les opérations implémentées :

- Possibilité d'accès à un élément quelconque.
- Parcours de la liste (dans un sens ou dans l'autre).
- Insertion et suppression aux extrémités de la liste.

Cependant, dans la représentation contigüe, les opérations d'insertion et de suppression à une position quelconque de la liste sont coûteuses (à cause des décalages), surtout si elles sont fréquentes.

Les Listes Chainées

Exemple :

Considérons la liste suivante des mots du dictionnaire anglais composés de 3 lettres et qui se terminent par ‘AT’.

L=(BAT, CAT, EAT, FAT, HAT, LAT, RAT).

La liste L est triée par ordre alphabétique (croissant). Il est évident que cette liste n’est pas complète. Nous pouvons constater :

- Qu’un autre mot peut être ajouté à la liste, par exemple le mot « GAT ».
- Qu’un des membres de la liste peut être supprimé, par exemple le mot « EAT » peut être supprimé si nous nous intéressons à une liste composée uniquement de noms (EAT est un verbe).

Les Listes Chainées

Supposons que la liste L est représentée par un tableau :

- L’insertion du mot « GAT » dans la liste nécessite le décalage d’une partie de la liste (HAT, LAT, RAT) pour obtenir la nouvelle liste L=(BAT, CAT, EAT, FAT, GAT, HAT, LAT, RAT).
- La suppression du mot « EAT » nécessite le décalage d’une partie de la liste (FAT, HAT, LAT, RAT).

Nous savons que ces décalages sont indésirables, en particulier pour :

- Des listes volumineuses.
- Des listes dans lesquelles les insertions et les suppressions sont très fréquentes.

Les Listes Chainées

Une solution possible serait :

- La représentation physique (en mémoire) ne respecte pas forcément l'ordre des éléments dans la liste.
- On associe à chaque élément de la liste une référence (un lien) vers l'élément suivant pour reconstituer l'ordre initial (ordre alphabétique).

Le parcours de la liste (en respectant l'ordre) ne consistera plus à parcourir dans l'ordre de représentation (ce qui est toujours le cas dans un tableau) mais en suivant les liens (le chainage).

Les Listes Chainées

2- Définition :

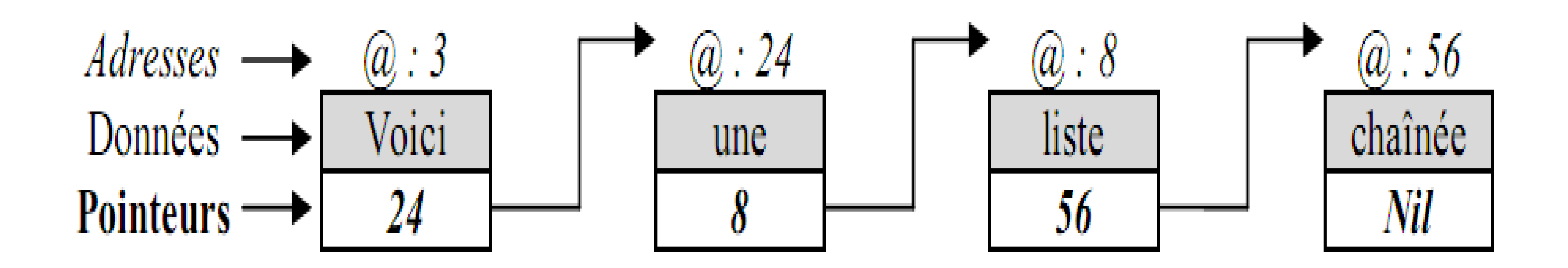
Une liste chaînée est une structure linéaire qui n'a pas de dimension fixée à sa création. Ses éléments de même type sont éparpillés dans la mémoire et reliés entre eux par des pointeurs. Sa dimension peut être modifiée selon la place disponible en mémoire. La liste est accessible uniquement par sa tête de liste c'est-à-dire son premier élément.

Pour les listes chaînées la séquence est mise en œuvre par le pointeur porté par chaque élément qui indique l'emplacement de l'élément suivant. Le dernier élément de la liste ne pointe sur rien (NULL).

On accède à un élément de la liste en parcourant les éléments grâce à leurs pointeurs.

Les Listes Chainées

Soit la liste chaînée suivante



Pour accéder au troisième élément de la liste il faut toujours débuter la lecture de la liste par son premier élément dans le pointeur duquel est indiqué la position du deuxième élément. Dans le pointeur du deuxième élément de la liste on trouve la position du troisième élément…

Les Listes Chainées

3- Utilisation des pointeurs pour le chaînage d’une liste

Chaque élément de la liste sera représenté par une structure (un maillon) contenant :

- Une donnée ou information,
- Un pointeur nommé Suivant indiquant la position de l'élément suivant dans la liste.

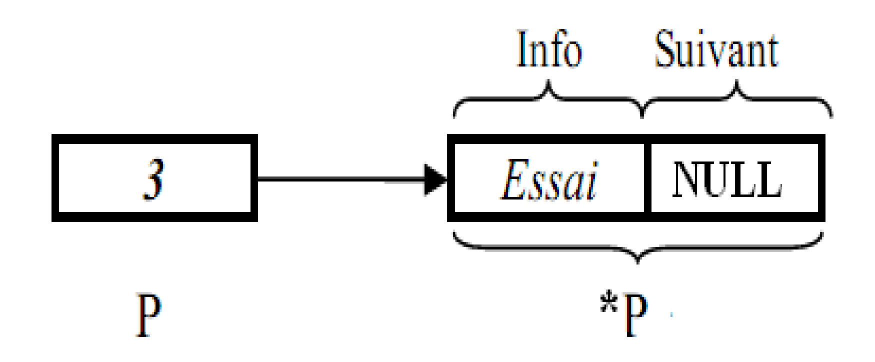
Tous les éléments d'une liste chaînée doivent avoir le même type

A chaque élément est associée une adresse mémoire.

Un pointeur est une variable dont la valeur est une adresse mémoire.
Un pointeur, noté P, pointe sur une variable dynamique notée *P.

Les Listes Chainées

Exemple



La variable pointeur **P** pointe sur l'espace mémoire **P^** d'adresse 3. Cette cellule mémoire contient la valeur "Essai" dans le champ Info et la valeur spéciale NULL dans le champ Suivant. Ce champ servira à indiquer quel est l'élément suivant lorsque la cellule fera partie d'une liste. La valeur NULL indique qu'il n'y a pas d'élément suivant. **P^** est l'objet dont l'adresse est rangée dans **P**.

Les Listes Chainées

Les listes chaînées entraînent l'utilisation de procédures d'allocation et de libération dynamiques de la mémoire. Ces procédures sont les suivantes:

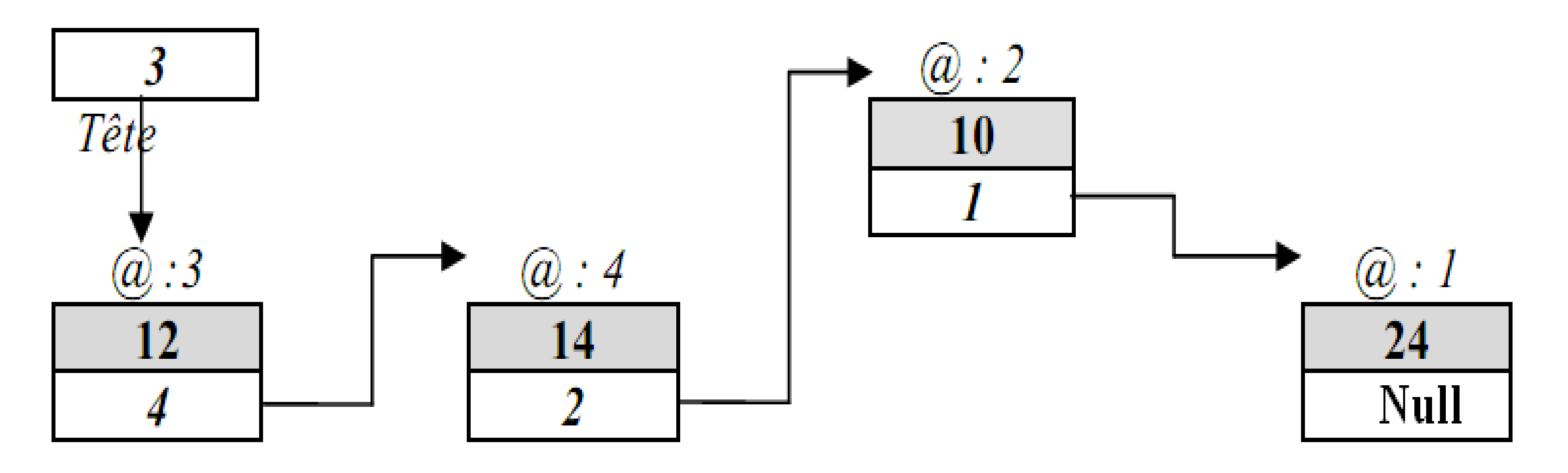
- New(P) : réserve un espace mémoire P^ et donne pour valeur à P l'adresse de cet espace mémoire. On alloue un espace mémoire pour un élément sur lequel pointe P.
- Dispose(P) : libère l'espace mémoire qui était occupé par l'élément à supprimer P^ sur lequel pointe P.

Pour pouvoir manipuler la liste nous avons besoin uniquement de connaitre l’adresse du première structure de la liste. Cette adresse sera conservée dans une variable «Tete » (pointeur de tête de liste).

La dernière structure ne doit pointer sur aucune autre structure. Pour cela la valeur spéciale NULL est utilisée pour marquer la fin de la liste.

Les Listes Chainées

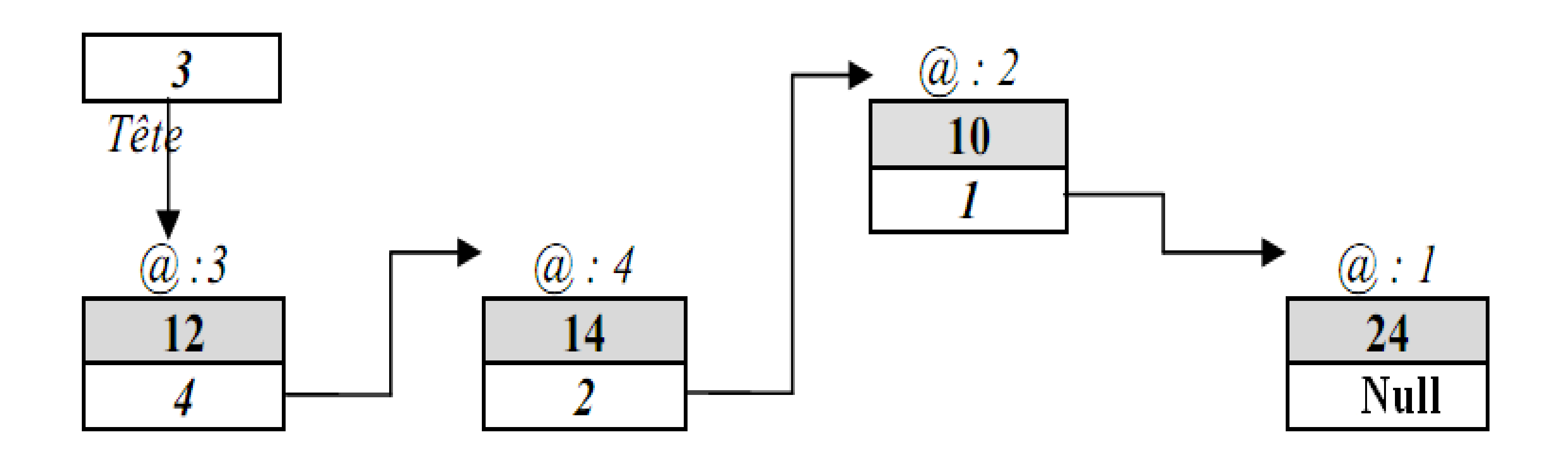
Avant d'écrire des algorithmes manipulant une liste chaînée, il est utile de montrer un schéma représentant graphiquement l'organisation des éléments de la liste chaînée.



Le premier élément de la liste vaut 12 à l'adresse 3 (début de la liste chaînée)

Le deuxième élément de la liste vaut 14 à l'adresse 4 (car le pointeur de la cellule d'adresse 3 est 4)

Les Listes Chainées



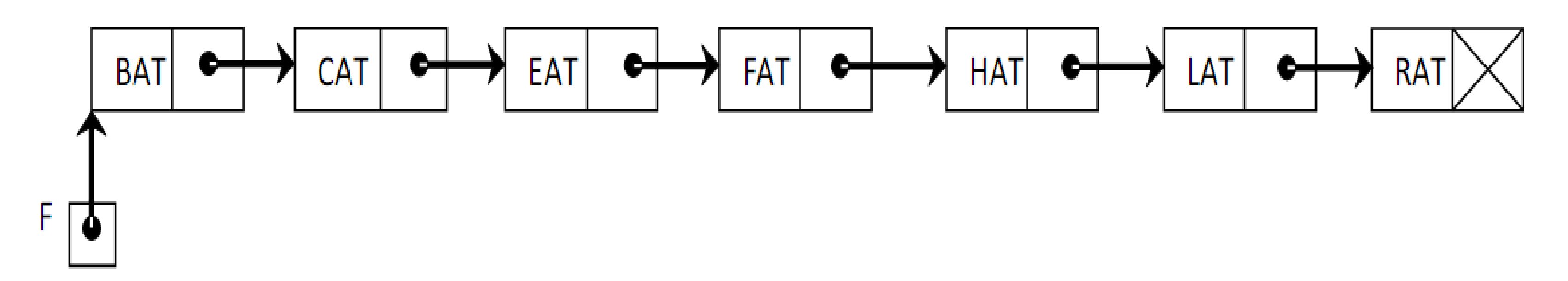
Le troisième élément de la liste vaut 10 à l'adresse 2 (car le pointeur de la cellule d'adresse 4 est 2)

Le quatrième élément de la liste vaut 24 à l'adresse 1 (car le pointeur de la cellule d'adresse 2 est 1)

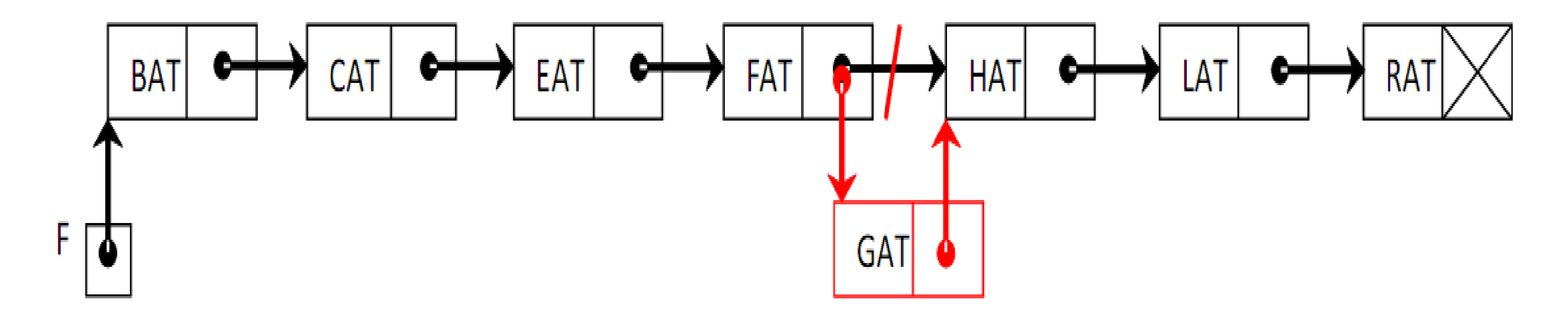
Les Listes Chainées

Exemple de la liste des mots du dictionnaire anglais

Schématiquement notre liste pourrait être représentée comme suit :

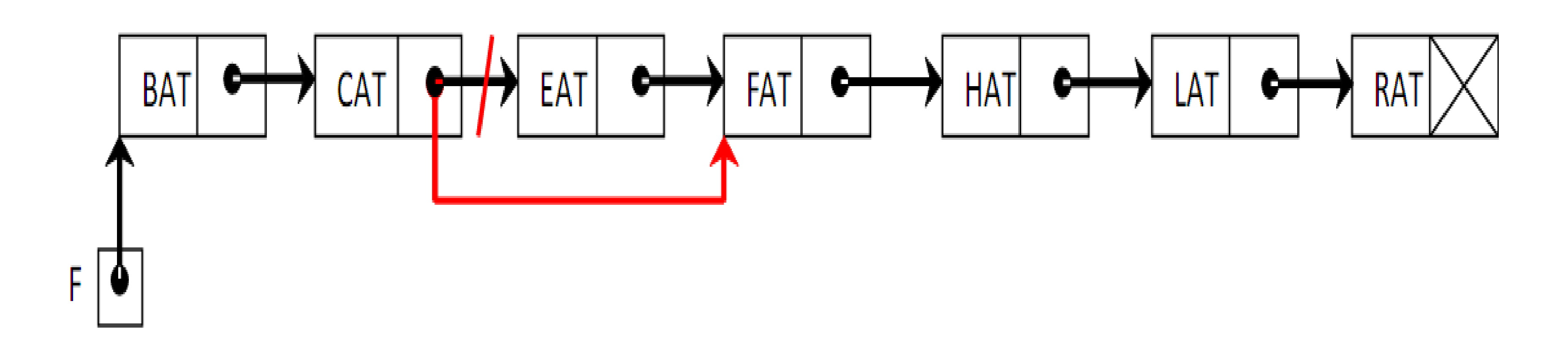


L'insertion d'un nouvel élément dans la liste peut se faire suivant le schéma suivant :



Les Listes Chainées

Et la suppression d'un élément (par exemple « EAT ») suivant ce schéma :



Les Listes Chainées

3.1) Déclarations

Soient **Liste** le type des éléments de la liste.

Les déclarations qui correspondent à la représentation proposée sont (si nous considérons l'exemple de liste de mots à 3 lettres) :

Notation inspirée du Pascal
Type Liste = enregistrement valeur : chaine[3] ; suivant : ^ liste ; **Fin structure** **Var** Tete, P : ^Liste

En langage C:

```
typedef struct Element
Element;
struct Element {
int nombre;
Element *suivant;
};
```

Les Listes Chainées

Le type « enregistrement Liste » correspond aux éléments qui composent notre liste. Il permet de déclarer des variables de type pointeurs sur cet enregistrement.

Les éléments d'une liste chaînée seront créés pendant l'exécution, c'est ce qu'on appelle une création dynamique.

Fonction **New(P)** renvoie un pointeur vers une nouvelle cellule allouée et l'action **Dispose(P)** récupère la cellule mémoire pointée par **P**.

Les Listes Chainées

Les traitements des listes sont les suivants :

- ❑ Créer et initialiser une liste.
- ❑ Ajouter un élément.
- ❑ Supprimer un élément.
- ❑ Modifier un élément.
- ❑ Afficher les éléments d'une liste.
- ❑ Rechercher une valeur dans une liste.

3.2) Initialisation de la liste

L'accès à la liste se fait à travers le pointeur de tête de liste « tete ». Il suffit donc de déclarer «tete» et de l'initialiser à la valeur spéciale NULL pour indiquer que la liste est vide (au départ).

tete : ^Liste;

tete←NULL ;

Les Listes Chainées

3.3)- Créer une liste chaînée composée de plusieurs éléments

Les éléments d'une liste chaînée seront créés pendant l'exécution, c'est ce qu'on appelle une création dynamique. Pour cela il faut :

- Déclarer une variable de type « ^Liste » pour contenir l'adresse d'élément à créer.
- Appeler une fonction d'allocation de mémoire, ***"New"***, pour réserver un espace mémoire correspondant à la taille d'un élément. Cette fonction retourne l'adresse de l'espace alloué.

Ce genre d'allocation s'appelle une allocation dynamique, par opposition à l'allocation statique pour les variables ordinaires (autres que les pointeurs). L'algorithme qui réalise cette action est :

Les Listes Chainées

L'accès à l'espace alloué se fait à travers la variable pointeur qui contient son adresse (dans l'exemple plus haut ″**p**″) en utilisant l'opérateur d'indirection ″^″. La variable ″p^″ désigne une variable de type ″Liste″ pointée par le pointeur ″p″.

```
Algorithme CréationListe
var Tete, P, L : ^Liste;
     Nbr_elt, cpt : entier;
Debut
lire (nbr_elt);
Tete ← Null;
New(P);        { réserve un espace mémoire pour l'élément à ajouter }
lire (P^.valeur);  {stocker dans le champ valeur de l'élément pointé par P}
P^.suivant ← Null;
Tete ← P;  { élément inséré en tete de la liste }
L ← P;
pour cpt ← 2 à nbr_elt faire
   New (P);
   lire (P^.valeur);
   P^.suivant ← Null;
   L^.suivant ← P;
   L ← P;
Fin pour;
FIN.
```

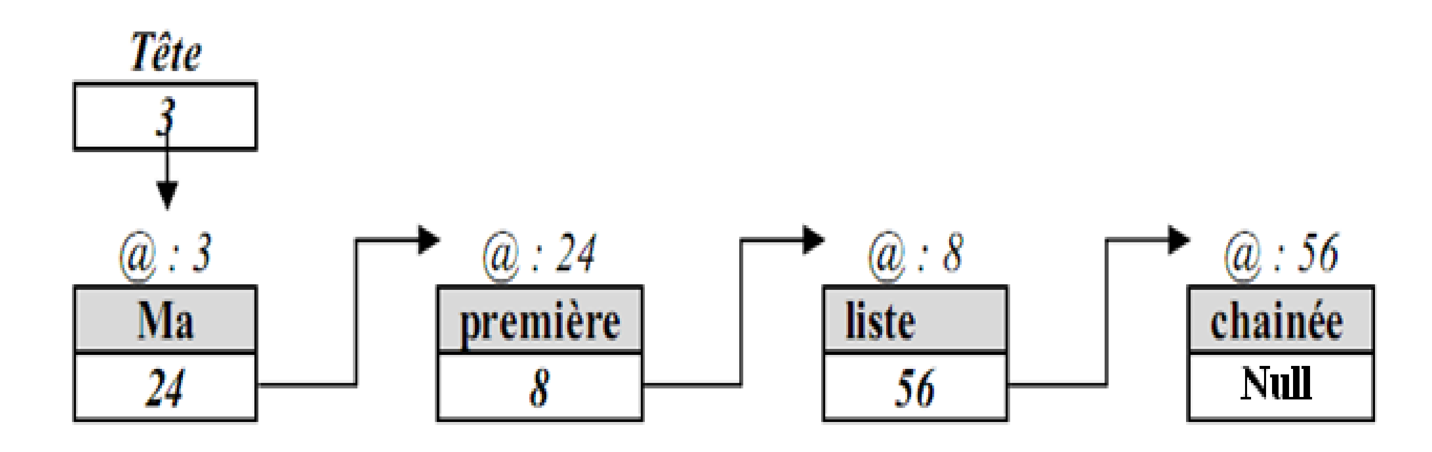
L'espace alloué n'est pas initialisé, il est donc incorrect de l'utiliser avant d'affecter aux différents champs de cet espace (élément) des valeurs valides.

Les Listes Chainées

3.4- Afficher les éléments d'une liste chaînée

Pour afficher les valeurs des éléments d'une liste, il est nécessaire de parcourir cette liste. Ce parcours nécessite la connaissance de l'adresse du premier élément de la liste (tête de liste). Cette adresse est stockée dans une variable « tete ».

Une liste chaînée simple ne peut être parcourue que du premier vers le dernier élément de la liste. Le parcours se fait comme suit :



Les Listes Chainées

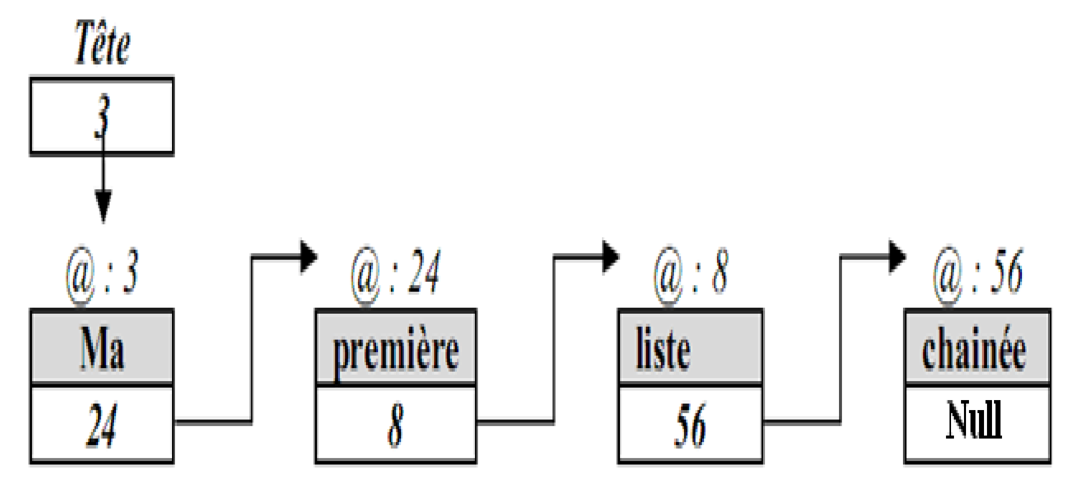
```
procedure afficher_liste (tete:^Liste)
var  P : ^Liste;

Debut
  P ← Tete   { P pointe sur le premier élément de la liste }
             {on parcourt la liste tant que l'adresse de l'élément suivant}
   Tant que P <> Null faire   {n'est pas Null}
     ecrire (P^.valeur);
     P ← P^.suivant;
   Fin tant que

FIN.
```

Si on applique la procédure pour cet exemple, l'affichage sera comme suit :

1	P a pour valeur 3
2	"Ma" s'affiche
3	P prend pour valeur 24
2	"première" s'affiche
3	P prend pour valeur 8
2	"liste" s'affiche
3	P prend pour valeur 56
2	"chaînée" s'affiche
3	P prend pour valeur Nil



Les Listes Chainées

3.5- Rechercher une valeur donnée dans une liste chaînée ordonnée

Dans cet exemple nous reprenons le cas de la liste chaînée contenant des éléments de type chaine de caractères, mais ce pourrait être tout autre type, selon celui déterminé à la création de la liste (rappelons que tous les éléments d'une liste chaînée doivent avoir le même type). La liste va être parcourue à partir de son premier élément (celui pointé par le pointeur de tête). Il a deux cas d’arrêt :

- avoir trouvé la valeur de l'élément,
- avoir atteint la fin de la liste.

L'algorithme est donné sous forme d'une procédure qui reçoit la tête de liste et la valeur à chercher en paramètre.

Les Listes Chainées

```
procedure Rechercher_valeur (tete:^Liste, val: chaine[3])
var  P : ^Liste;
       existe : booléen;
Debut
  Si tete <> Null alors
    P ← Tete   { P pointe sur le premier élément de la liste }
    existe ← faux;
 {on parcourt la liste tant que l'adresse de l'élément suivant n'est pas Null}
     Tant que P <> Null et non existe faire
        Si  P^.valeur = val alors
           existe  ← vrai
        Sinon
           P ← P^.suivant;
        Fin Si;
     Fin tant que
     Si existe = vrai alors
        ecrire ('la valeur',val,'est dans la liste')
     Sinon
        ecrire ('la valeur',val,'n est pas dans la liste');
     Fin Si
   sinon  ecrire ('la liste est vide');
   Fin Si;
FIN.
```